AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Informatyka Stosowana

DOKUMENTACJA PROJEKTOWA

Inżynieria Oprogramowania

Aplikacja Quizowa "BrainCell"

Autorzy:
Dariusz Łopian
Motyka Szymon
Matras Szymon

Prowadzący:
mgr inż. Daniel Drozd

Nowy Sącz 2022

Spis treści

1. Opis i cel projektu

Aplikacja **BrainCell** to gra mobilna oparta o aplikacje typu quiz, w którym mamy za zadanie wybrać jedną prawidłową odpowiedź z czterech podanych. Dzięki temu, że aplikacja będzie napisana na telefony komórkowe z systemem android, uzyskamy możliwość sprawdzenia swojego poziomu wiedzy w dowolnym miejscu. Użytkownik po zalogowaniu będzie miał dostęp do podglądu kategorii w jakich chce odpowiadać na pytania a przez to baza pytań będzie zawężona. Gra będzie skierowana do wszystkich użytkowników zarówno młodych, jak i starszych, ponieważ poziom pytań będzie bardzo zróżnicowany. Interfejs i obsługa będzie prosta i intuicyjna dzieki czemu nikt nie bedzię miał problemu z poruszaniem się po menu oraz po samej rozgrywce.
Po wejściu do aplikacji naszym oczom ukaże się ekran logowania. Po poprawnym zalogowaniu uzyskamy widok głównego menu, w którym znajdziemy:
- Profil użytkownika - z poziomu którego będzie można dokonywać edycji naszych danych
- Wybór kategorii pytań
- Osiągnięcia
- Ustawienia

Rozgrywka będzię kontrolowana przez czas. Po wybraniu interesującej nas kategorii ujrzymy pytanie oraz cztery odpowiedzi, z których jedna będzie poprawna. W górnej części ekranu zostanie wyświetlony timer, który będzie odliczał czas. Po upływie ustalonego czasu i nie zaznaczeniu żadnej odpowiedzi przez użytkownika, zostanie ona uznana jako błędna. Następnie na ekranie zostanie wyświetlone kolejne pytanie. Szata graficzna aplikacji będzie prosta, czytelna i schludna, dzieki czemu rozgrywka będzie przyjemna.

2. Wymagania funkcjonalne i niefunkcjonalne

a) Wymagania funkcjonalne:
- Możliwość utworzenia konta i zalogowania się do aplikacji
- Możliwość edycji danych w profilu użytkownika
- Możliwość wyboru kategorii pytań
- Możliwość wyboru i zaznaczenie odpowiedzi na zadane pytanie
- Możliwość sprawdzenia poprawnej odpowiedzi
- Możliwość weryfikacji swoich osiągnięć w grze
- Możliwość zmiany motywu aplikacji w ustawieniach (??)

b) Wymagania niefunkcjonalne:
Po wdrożeniu większości powyższych wymagań funkcjonalnych, powinniśmy otrzymać wymagania niefunkcjonalne, takie jak:
- Przejrzysta i responsywna aplikacja mobilna dla użytkowników
- Czytelny i prosty w obsłudze interfejs

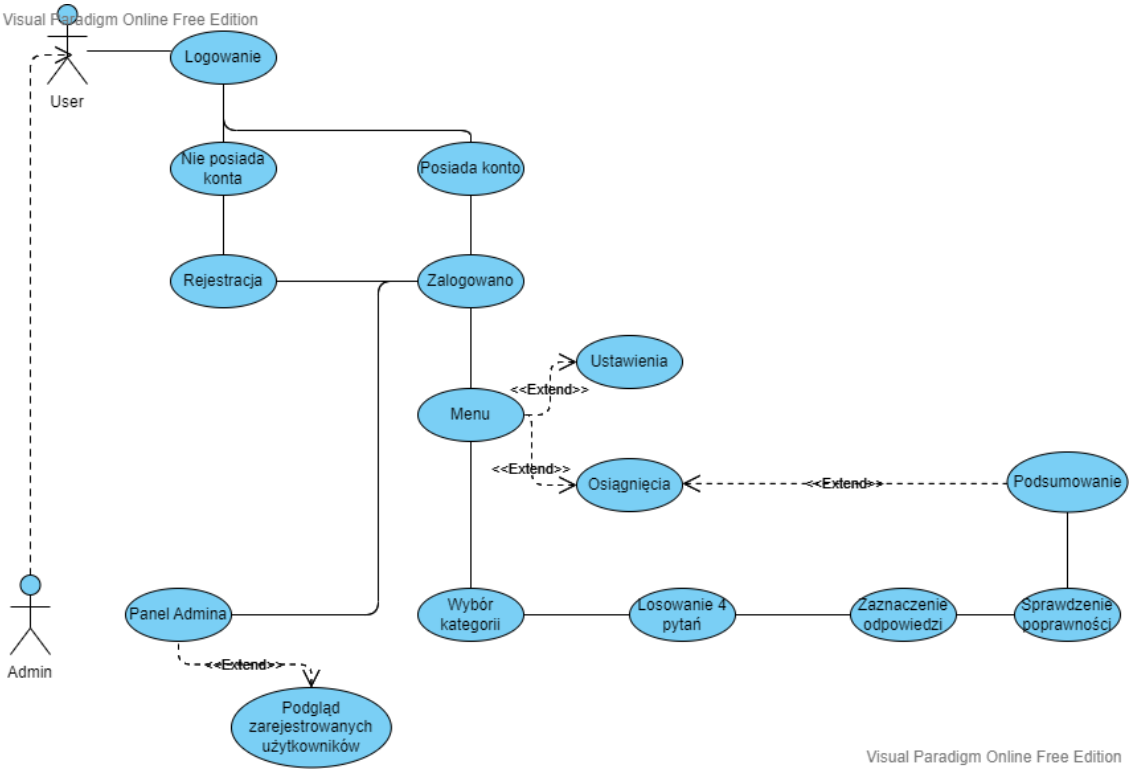
3. Opis technologii

Podczas realizacji projektu wykorzystamy takie technologie jak:
- ***Flutter*** - jest to framework stworzony przez Google, za pomocą którego można tworzyć aplikacje mobilne na urządzenia z systemem Android oraz iOS. Narzędzie to rozwija się dość dynamicznie, ponieważ skupia wokół siebie dużą część społeczności programistów, dzięki łatwej i przyspieszonej pracy, a także pozwala osiągnąć zamierzone efekty przy zaangażowaniu mniejszej liczby programistów i niższych kosztach.
- ***C++*** - jest to jeden z bardziej popularnych języków programowania ogólnego przeznaczenia. Stosuje różne style programowania, dzięki czemu nadaje się do tworzenia systemów operacyjnych, systemów wbudowanych, aplikacji desktopowych, serwerów, czy silników gier.
- ***Firebase*** - należąca do Google platforma umożliwiająca tworzenie aplikacji mobilnych oraz internetowych, wyposażona dodatkowo w bazę danych.

4. Identyfikacja problemów oraz proponowane rozwiązania

Naszym największym problemem do rozwiązania przy tworzeniu aplikacji będzie stworzenie bazy pytań wraz z odpowiadającymi im kategoriami. Rozwiązaniem problemu może okazać się skorzystanie z gotowej już bazy pytań.

5. Diagram przypadków użycia



Rys. 5.1. Diagram przypadków użycia 'BrainCell'

6. Scenariusze przypadków użycia

a) Scenariusze użytkownika.

Scenariusz nr 1

Tytuł: Logowanie

Warunek wejścia: Posiada konto

Przebieg: Użytkownik podejmuje próbę logowania. Podaje niezbędne dane (login, hasło).

Zakończenie: Jeśli próba logowania jest pozytywna - zyskuje dostęp do pełnej funkcjonalności aplikacji, w przeciwnym razie dostaje kolejną możliwość wprowadzenia danych logowania.

Zakończenie alternatywne: Jeśli użytkownik nie posiada konta dostanie możliwość wykonania scenariusza nr 2 (Rejestracja).

Scenariusz nr 2,

Tytuł: Rejestracja

Warunek wejścia: Musi wejść do aplikacji (kliknąć w ikonę aplikacji).

Przebieg: Użytkownik dostaje możliwość stworzenia swojego profilu, który jest konieczny do skorzystania z aplikacji.

Zakończenie: Jeśli rejestracja przebiegnie pomyślnie użytkownik dostanie możliwość zalogowania się do aplikacji.

Zakończenie alternatywne: W przypadku podania błędnych danych nastąpi wyświetlenie komunikatu błędu i możliwość podjęcia kolejnej próby.

Scenariusz nr 3

Tytuł: Sprawdzenie osiągnięć

Warunek wejścia: Użytkownik jest zalogowany i znajduje się w głównym menu.

Przebieg: Użytkownik w menu głównym klika w Osiągnięcia.

Zakończenie: Po kliknięciu następuje przejście do widoku osiągnięć, gdzie użytkownik ma możliwość sprawdzenia swoich statystyk z odbytych gier.

Scenariusz nr 4

Tytuł: Wybór kategorii pytań

Warunek wejścia: Użytkownik jest zalogowany i znajduje się w głównym menu.

Przebieg: Użytkownik w menu głównym klika w Kategorie pytań.

Zakończenie: Po kliknięciu następuje przejście do widoku kategorii pytań z danej dziedziny wiedzy, gdzie użytkownik wybiera kategorie z jakiej chce odpowiadać.

Scenariusz nr 5

Tytuł: Gra - odpowiadanie na pytania

Warunek wejścia: Użytkownik wybrał kategorie z jakiej chce odpowiadać.

Przebieg: Po wybraniu interesującej kategorii, rozpoczyna się gra, w której użytkownikowi ukazuje się losowe pytanie z czterema odpowiedziami do wyboru.

Zakończenie: Użytkownik wybiera jego zdaniem poprawną odpowiedź, a następnie następuje weryfikacja zaznaczonej odpowiedzi.

Zakończenie alternatywne: Może zdarzyć się, że użytkownik nie zaznaczy żadnej z podanych odpowiedzi, wtedy po upływie ustalonego czasu nastąpi weryfikacja odpowiedzi.

Scenariusz nr 6

Tytuł: Weryfikacja odpowiedzi

Warunek wejścia: Użytkownik zaznaczył odpowiedź, bądź nie.

Przebieg: Po wybraniu odpowiedzi, czy też upływie czasu - automatycznie przez system zaznaczana jest prawidłowa odpowiedź.

Zakończenie: Po weryfikacji generowane jest kolejne pytanie z puli.

Scenariusz nr 7

Tytuł: Podsumowanie wyniku i zakończenie gry.

Warunek wejścia: Wyczerpanie się puli pytań

Przebieg: Po przejściu 10 pytań z danej kategorii oraz weryfikacji ich poprawności następuje koniec gry, po której użytkownikowi wyświetlają się statystyki za rozgrywkę.

Zakończenie: Po kliknięciu odpowiedniego przycisku następuje przeniesienie do głównego menu, a zdobyte punkty aktualizują się w osiągnięciach profilu.

Scenariusz nr 8

Tytuł: Przerwanie gry.

Warunek wejścia: Użytkownik znajduje się w trybie gry.

Przebieg: Użytkownik podczas rozgrywki ma możliwość przerwania jej w każdej chwili poprzez kliknięcie odpowiedniego przycisku. Gdy to zrobi, gra automatycznie się zakończy, a on znajdzie się w głównym menu.

Zakończenie: Po kliknięciu odpowiedniego przycisku następuje przeniesienie do głównego menu, a zdobyte punkty nie ulegają zapisaniu.

Scenariusz nr 9

Tytuł: Wyjście z aplikacji.

Warunek wejścia: Użytkownik znajduje się w menu głównym.

Przebieg: Użytkownik będąc w głównym menu klika przycisk "WYJDŹ"

Zakończenie: Po kliknięciu następuje zamknięcie aplikacji.

b) Scenariusze administratora.

Scenariusz nr 1

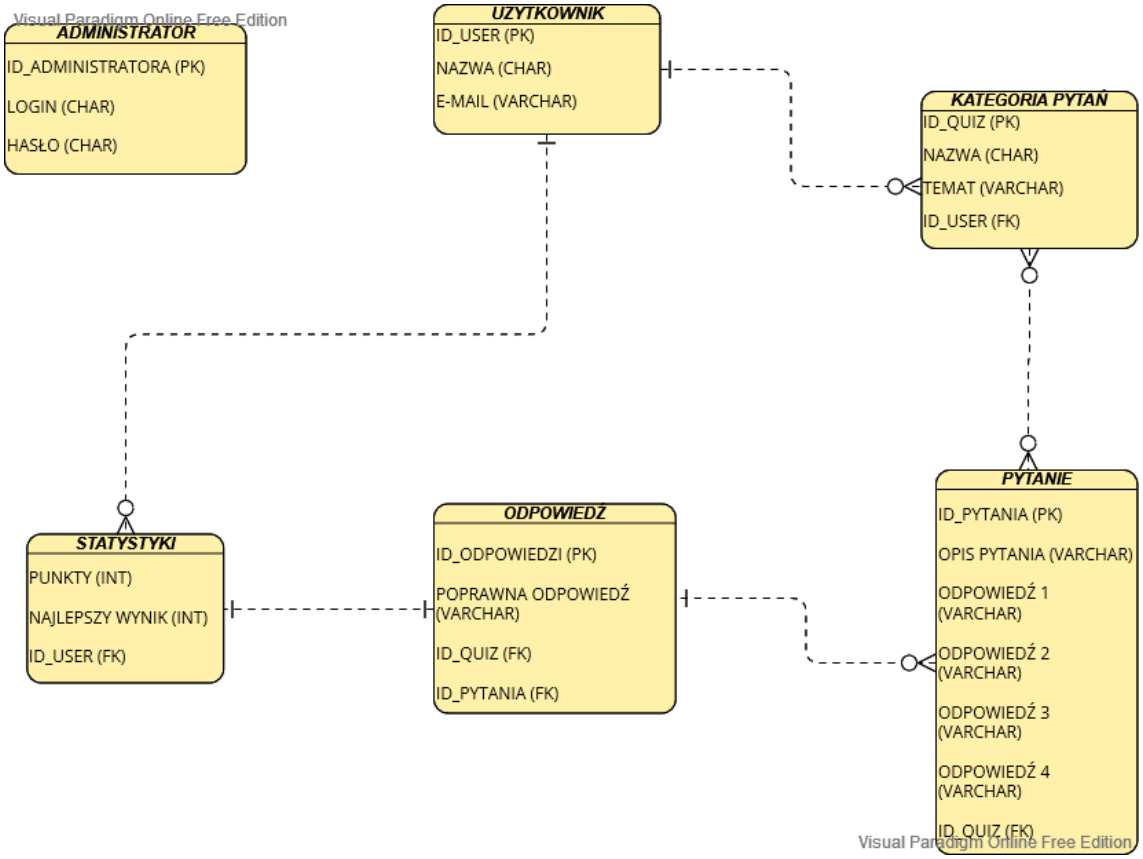
Tytuł: Logowanie

Warunek wejścia: Konto posiada uprawnienia administratora.

Przebieg: Administrator loguje się do systemu, dzięki czemu zyskuje dostęp do funkcji administratorskich.

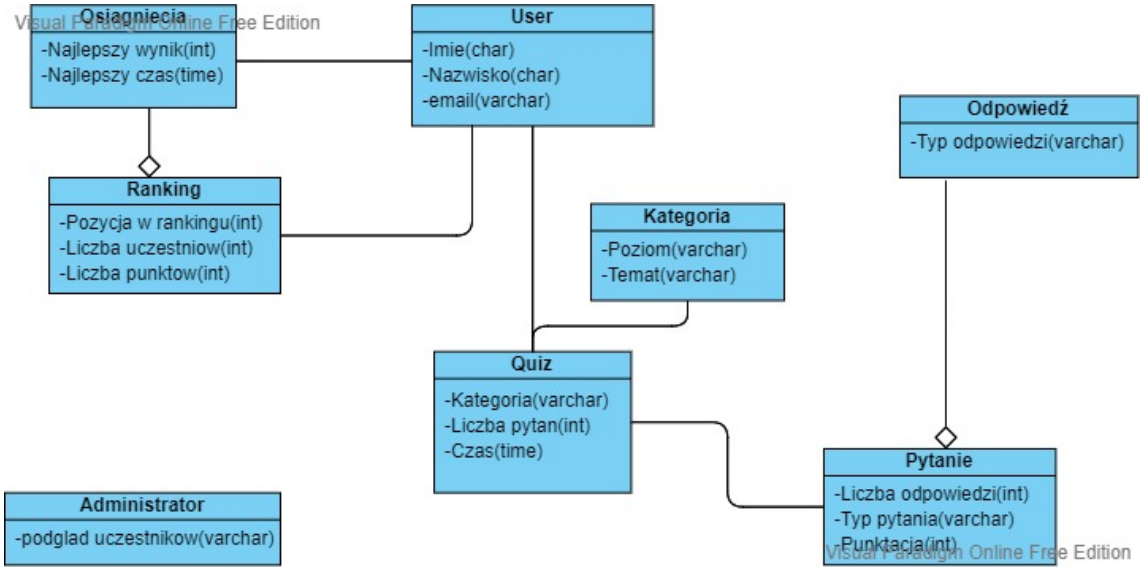
Zakończenie: Zalogowanie do konta oraz możliwość zarządzania (m. in. podgląd listy użytkowników).

7. Diagram ERD - Entity Relationship Database



Rys. 7.1. Diagram ERD 'BrainCell'

8. Diagram Klas



Rys. 8.1. Diagram Klas 'BrainCell'

9. Rozpoznanie wzorców projektowych

a) Metoda szablonowa - to wzorzec projektowy definiujący szkielet algorytmu w klasie bazowej. Pozwala podklasom nadpisać pewne etapy algorytmu bez konieczności zmiany ich struktury.
Rozwiązanie zawarte we wzorcu Metody szablonowej zakłada rozdzielenie algorytmu na kolejne etapy, utworzenie z tych etapów metod i umieszczenie ciągu wywołań poszczególnych metod w jednej metodzie szablonowej. Etapy mogą być albo abstrakcyjne, albo posiadać jakąś domyślną implementację. Aby skorzystać z algorytmu, klient powinien dostarczyć swoją podklasę implementującą wszystkie etapy abstrakcyjne i nadpisać opcjonalne etapy jeśli jest taka potrzeba (oprócz samej metody szablonowej).
b) Budowniczy - jest kreacyjnym wzorcem projektowym, który daje możliwość tworzenia złożonych obiektów etapami, krok po kroku. Wzorzec ten pozwala produkować różne typy oraz reprezentacje obiektu używając tego samego kodu konstrukcyjnego. Wzorzec ten proponuje ekstrakcję kodu konstrukcyjnego obiektu z jego klasy i umieszczenie go w osobnych obiektach zwanych budowniczymi. Dzieli on konstrukcję obiektu na pewne etapy. Aby powołać do życia obiekt, wykonuje się ciąg takich etapów za pośrednictwem obiektu-budowniczego. Istotne jest to, że nie trzeba wywoływać wszystkich etapów naraz. Można bowiem ograniczyć się tylko do tych kroków, które są niezbędne do określenia potrzebnej nam konfiguracji obiektu na dany moment.
c) Adapter - jest strukturalnym wzorcem projektowym pozwalającym na współdziałanie ze sobą obiektów o niekompatybilnych interfejsach. Jest to specjalny obiekt konwertujący interfejs jednego z obiektów w taki sposób, że drugi obiekt go rozumie. Adapter stanowi swego rodzaju opakowanie dla obiektu, ukrywając szczegóły konwersji jakie odbywają się za kulisami. Obiekt opakowywany może nawet nie wiedzieć o istnieniu adaptera. Można na przykład opakować obiekt korzystający z jednostek kilometr i metr w adapter konwertujący te dane na jednostki imperialne, takie jak stopy i mile. Adaptery mogą nie tylko konwertować dane pomiędzy formatami, ale również pozwolić na współpracę obiektów o różnych interfejsach. Działa to tak:
Adapter uzyskuje interfejs kompatybilny z interfejsem jednego z obiektów. Następnie za pomocą tego interfejsu, istniejący obiekt może śmiało wywoływać metody adaptera. Otrzymawszy wywołanie, adapter przekazuje je dalej, ale już w formacie obsługiwanym przez opakowany obiekt.
Czasami jest nawet możliwe stworzenie adaptera dwukierunkowego, potrafiącego konwertować wywołania w obu kierunkach.

d) Singleton - jest kreacyjnym wzorcem projektowym, który pozwala zapewnić istnienie wyłącznie jednej instancji danej klasy. Ponadto daje globalny punkt dostępowy do tejże instancji. Singleton rozwiązuje jednocześnie dwa problemy. Po pierwsze zapewnia istnienie wyłącznie jednej instancji danej klasy, a po drugie pozwala na dostęp do tej instancji w przestrzeni globalnej.
Wszystkie implementacje wzorca Singleton współdzielą poniższe dwa etapy:
Ograniczenie dostępu do domyślnego konstruktora przez uczynienie go prywatnym, aby zapobiec stosowaniu operatora new w stosunku do klasy Singleton.
Utworzenie statycznej metody kreacyjnej, która będzie pełniła rolę konstruktora. Za kulisami, metoda ta wywoła prywatny konstruktor, aby utworzyć instancję obiektu i umieści go w polu statycznym klasy. Wszystkie kolejne wywołania tej metody zwrócą już istniejący obiekt.
e) Odwiedzający - to behawioralny wzorzec projektowy pozwalający oddzielić algorytmy od obiektów na których pracują.
Wzorzec projektowy Odwiedzający proponuje umieszczenie nowych obowiązków w osobnej klasie zwanej odwiedzającym, zamiast próbować zintegrować je z istniejącymi klasami. Pierwotny obiekt, który miał wykonywać te obowiązki, teraz jest przekazywany do jednej z metod odwiedzającego w charakterze argumentu. Daje to metodzie dostęp do wszystkich potrzebnych danych znajdujących się w obiekcie.
f) Dekorator - umożliwia dynamiczne przydzielenie wybranemu obiektowi nowych zachowań. Dekoratory dają elastyczność podobną do tej, jaką daje dziedziczenie, jednak oferują znacznie lepszą funkcjonalność.
g) Fasada - należy do wzorców strukturalnych. W tym wzorcu jedna klasa nadrzędna zawiera mniejsze obiekty klas, którymi zarządza udostępniając użytkownikowi (programowi) zewnętrznemu odpowiednie metody upraszczające obsługę działań na obiektach weń znajdujących się.
g) Kompozyt - umożliwia on tworzenie struktur drzewiastych, gdzie podstawową jednostką jest liść a rozszerzoną kompozyt, przy czym kompozyt można opisać jako swego rodzaju gałąź struktury. Oba te elementy (liść i kompozyt) dziedziczą po wspólnym interfejsie. Sam kompozyt zawiera kontener, który może przechowywać interfejsy, za którymi mogą stać jednostki podstawowe (liście) lub kolejne gałęzie (kompozyty).
h) Obserwator - jego celem jest wywoływanie klasy zawierającej klasy obserwatorów w momencie, gdy zajdzie określona zmiana w obiekcie obserwowanym. Innymi słowy jeżeli obserwowany obiekt zmienia stan, wtedy uruchamia on odpowiednią metodę klasy zawierającej obserwatorów a ta wywołuje kolejno metody na zarejestrowanych obserwatorach.

W naszym projekcie został użyty wzorzec projektowy o nazwie bloc. Jest to bardzo popularny wzorzec w środowisku programistycznym flutter i jest często wykorzystywany przez duże korporacje. Wzorzec bloc służy do oddzielenia warstwy logicznej - backendu od warstwy biznesowej - UI/layout. Bloc pozwala na wygodne ustrukturyzowanie plików w projekcie, tak aby wiele osób mogło pracować skutecznie w ramach jednego projektu. Bloc również rozwiązuje wiele podstawowych problemów związanych z state managment, co pozwala na zarządzaniu stanami za pośrednictwem innych klas. Bloc zwykle składa się z modeli, czyli klas na których będziemy operować, view czyli widoków dla poszczególnych bloków, eventów, czyli akcji, które będą wywoływały jakieś akcje np. zmiany stanów oraz stany jakie może przyjmować dany blok. We flutterze aby bloc działał poprawnie, należy zaimportować paczkę bloc oraz użyć blocbuilder'a, aby przechodzić pomiędzy stanami wywołując dane widoki za pomocą eventów.

```
13  class QueryBloc extends Bloc<QueryEvent, QueryState> {
14    late List<Query> list;
15    int idx = 0;
16    int pointScored = 0;
17    int myAnswer = 0;
18    List<int> myAnswers = [];
19
20    QueryBloc() : super(QueryInitial()) {
21      on<LoadEvent>((event, emit) async {
22        emit(QueryShimmerState());
23        list = event.list;
24        await Future.delayed(const Duration(seconds: 1));
25        emit(QueryChanged(list.first));
26      });
27
28      on<QueryChangeEvent>((event, emit) async {
29        emit(QueryLoadingState(list[idx]));
30        myAnswers.add(event.myAnswer);
31        if (event.myAnswer == state.query.correctAnswer) {
32          pointScored++;
33        }
34        myAnswer = event.myAnswer;
35        idx++;
36        await Future.delayed(const Duration(seconds: 1));
37        if (idx < list.length) {
38          emit(QueryChanged(list[idx]));
39        }
40      });
41    }
42  }
```

Rys. 9.1. Bloc - implementacja

Spis rysunków